АНОТАЦІЯ

«Розробка системи автоматизованого тестування з використанням інструментів Selenium і Jenkins та середовища Intellij IDEA» // Мельник Андрій Володимирович // Тернопільський національний технічний університет імені Івана Пулюя, факультет комп'ютерно-інформаційних систем і програмної інженерії, кафедра інженерії програмного забезпечення, група СПм-61 // Тернопіль, 2023 // С. 55, табл. 0 , рис. 16 , додат. – 1.

Ключові слова: ТЕСТУВАННЯ, АВТОМАТИЗАЦІЯ, ТЕСТОВИЙ СЦЕНАРІЙ, SELENIUM, JAVA, JENKINS, TESTNG, INTELLIJ IDEA, PAGE OBJECT, CONTINUOUS INTEGRATION, GRADLE, EXTENT REPORT.

Метою даної магістерської роботи є розробка системи автоматизованого тестування з використанням популярних інструментів Selenium і Jenkins у поєднанні з інтегрованим середовищем розробки Intellij IDEA. Робота спрямована на вдосконалення процесу тестування програмного забезпечення, забезпечуючи ефективне використання інструментів для автоматизації тестів та їхню інтеграцію в процес розробки за допомогою системи Continuous Integration (CI).

ANNOTATION

«Development of an automated testing system using Selenium and Jenkins tools and the Intellij IDEA environment» // Master’s Thessis // Melnyk Andrii // Ternopil Ivan Puluj National Technical University, Faculty of Computer Information System and Software Engineering, Department of Cybersecurity // Ternopil, 2023 // P. 55, Tables 0 , Fig. 16 , Annexes – 1.

Keywords: TESTING, AUTOMATION, TEST SCRIPT, SELENIUM, JAVA, JENKINS, TESTNG, INTELLIJ IDEA, PAGE OBJECT, CONTINUOUS INTEGRATION, EXTENT REPORT.

The purpose of this master's thesis is to develop an automated testing system using the popular tools Selenium and Jenkins in combination with the integrated development environment Intellij IDEA. The work is aimed at improving the software testing process, ensuring the effective use of test automation tools and their integration into the development process using the Continuous Integration (CI) system.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

CI - Continuous Integration

CD - Continuous Delivery

IDE - Integrated Development Environment

API - Application Programming Interface

AI - Artificial intelligence

HTML - Hypertext Markup Language

XML - Extensible Markup Language

POM - Project Object Model

NG – Next Generation

UI – User Interface

ВСТУП

Актуальність теми. Сучасна індустрія розробки програмного забезпечення вимагає високої якості, швидкості та надійності продуктів. Однією з ключових складових успіху є ефективний та автоматизований процес тестування, який дозволяє вчасно виявляти та виправляти помилки, забезпечуючи високий рівень функціональності та стабільності програм. З кожним днем сучасний інтернет відіграє все більш визначну роль у житті людей та бізнес-середовищі, внаслідок чого веб-додатки стають необхіднішими і складнішими.

Однак, разом із зростанням складності веб-додатків збільшується ймовірність виникнення помилок та дефектів у їхньому функціонуванні. Для забезпечення стабільності та надійності веб-додатків, а також швидкого впровадження нових функцій та змін, важливо мати ефективну систему тестування.

Веб-додатки повинні працювати в різних браузерах, пристроях та операційних системах. Автоматизовані тести дозволяють впевнено перевіряти сумісність програмного забезпечення на різних платформах. Також вони дозволяють швидше виявляти помилки та автоматично виконувати тести, що раніше вимагало б значно більше зусиль при ручному тестуванні. Це покращує продуктивність розробників та забезпечує більш ефективне використання робочого часу.

Системи автоматизованого тестування стають невід'ємною частиною розробки веб-додатків. Автоматизація тестування дозволяє швидко та ефективно перевіряти функціональність, а також забезпечує підтримку високої якості продукту протягом усього циклу розробки. Автоматизовані тести дозволяють виконувати тестування швидше та ефективніше, зменшуючи ризик людських помилок та витрат часу на рутинні завдання.

Зростаюча конкуренція на ринку вимагає швидкого впровадження нових можливостей та регулярні оновлення веб-додатків. Системи автоматизованого тестування допомагають скоротити час розробки та випуску нових версій, що є критичним для успіху в сучасному бізнес-середовищі. Таким чином, розробка системи автоматизованого тестування для веб-сторінок є актуальною та стратегічно важливою задачею в галузі програмної інженерії.

Усі ці фактори свідчать про необхідність та актуальність впровадження систем автоматизованого тестування для веб-сторінок, які допомагають забезпечити якість, надійність та швидкість випуску програмного забезпечення в умовах постійних змін та вимог ринку.

Мета даного дослідження – визначити ефективність та практичну застосовність обраного підходу до автоматизованого тестування та інтеграції в процес розробки програмного забезпечення. Робота базується на практичних дослідженнях та застосуванні системи автоматизованого тестування на реальному проекті.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Поняття автоматизованого тестування

Автоматизоване тестування - це процес використання програмних інструментів для виконання тестів на програмному продукті. Це включає в себе виконання тестових сценаріїв, порівняння фактичних результатів з очікуваними та автоматизацію процесів тестування [1]. Автоматизоване тестування використовує програмні скрипти для відтворення конкретних тестових сценаріїв. Це може включати в себе введення користувача, натискання кнопок, перевірку виведення на екрані та багато іншого.

Існує багато інструментів для автоматизованого тестування, які допомагають у написанні, виконанні та аналізі автоматизованих тестів. Деякі з них включають Selenium для веб-додатків, Appium для мобільних додатків, JUnit та TestNG для тестування Java-програм та багато інших.

Однією з ключових частин автоматизованого тестування є написання тестових скриптів. Це може вимагати програмування для створення автоматизованих тестів, які виконують певні дії та перевіряють очікувані результати. Автоматизоване тестування часто використовується для регресійного тестування, яке спрямоване на виявлення нових помилок, які можуть з'явитися при внесенні змін у вихідний код програми.

Автоматизоване тестування інтегрується в процес безперервної інтеграції(continuous integration), де тести виконуються автоматично при кожній зміні в коді. Це допомагає виявляти помилки раніше та підтримувати стабільність системи. Також автоматизоване тестування може бути використане для тестування різних аспектів програмного продукту, таких як функціональність, продуктивність, безпека та інші.

1.2 Для чого використовується автоматизація тестування

Автоматизоване тестування використовується для автоматизації виконання тестових сценаріїв і перевірки різноманітних аспектів програмного продукту. Автоматизація дозволяє виконувати тести значно швидше, ніж це може бути зроблено вручну. Це особливо важливо в умовах постійного розвитку та інтеграції продукту.

Тести можна легко повторювати при кожній новій версії програми чи при змінах в коді. Це допомагає виявляти помилки та проблеми, які можуть виникнути під час розробки.

Хоча розробка автоматизованих тестів вимагає певних витрат, вони часто економлять час та ресурси в подальшому. Після написання автоматизованих тестів їх можна використовувати багаторазово без значних витрат.

Автоматизовані тести можуть виконувати однакові дії з високою точністю, уникнувши тих помилок, які можуть бути зумовлені втомою чи невірним виконанням інструкцій. Завдяки автоматизації можна швидко виконувати тести при різних умовах, таких як різні конфігурації операційних систем, браузерів або мережевих умов.

Автоматизовані тести допомагають швидко виявляти регресійні помилки, тобто проблеми, що виникають під час розробки нового функціоналу або зміни в програмі, які можуть впливати на існуючий код. Автоматизоване тестування дозволяє вбудовувати тести в процес неперервної інтеграції, де новий код автоматично перевіряється на відповідність стандартам якості перед включенням у основний код.

Використання автоматизованого тестування сприяє покращенню якості програмного продукту, зменшенню часу розробки і випуску нових версій, а також зниженню витрат на тестування у довгостроковій перспективі.

Одним з викликів автоматизованого тестування - це правильний вибір тестових сценаріїв для автоматизації. Не всі сценарії є ефективними для автоматизації, і необхідно визначити, які саме тести стануть кандидатами для автоматизації. Також підтримка тестових скриптів може стати важким завданням, особливо у випадку частих змін у програмному забезпеченні. Важливо забезпечити сталу актуальність тестових скриптів та їхню відповідність змінам у системі.

1.3 Аналіз сучасних підходів до автоматизації тестування

Сучасне програмування вимагає від розробників та тестувальників використання різноманітних інструментів та стратегій для автоматизації тестування. Один із підходів, що використовується для функціонального тестування, це "Record and Playback". Цей метод дозволяє швидко створювати тести, але він може бути вразливим до змін в інтерфейсі програми. З іншого боку, скриптоване тестування (за допомогою інструментів, таких як Selenium чи Appium) надає більший контроль та гнучкість, але вимагає більше часу для розробки.

У сфері навантажувального тестування інструменти, такі як Apache JMeter чи Gatling, дозволяють моделювати реальне навантаження та оцінювати продуктивність системи. Це особливо важливо для великих та високонавантажених додатків.

Тестування API включає в себе юніт-тестування та інтеграційне тестування. Юніт-тестування спрямоване на перевірку окремих компонентів програми, тоді як інтеграційне тестування фокусується на взаємодії між компонентами та системами.

Хмарні інструменти, такі як Sauce Labs чи BrowserStack, надають можливість тестувати програмне забезпечення на різних пристроях та в різних браузерах в хмарному середовищі, забезпечуючи максимальну покриття.

Масштабовані тестувальні платформи, такі як TestRail чи TestLink, допомагають ефективно керувати тестовими випадками та результатами, сприяючи організації тестових процесів.

Інструменти для обробки даних та аналізу результатів, наприклад ExtentReports, грають важливу роль у створенні інформативних звітів, які сприяють аналізу результатів тестування.

На сьогоднішній день спостерігається також загальний тренд до інтеграції інтелектуальних технологій в процеси тестування. Використання штучного інтелекту в тестуванні (AI Testing) включає в себе використання алгоритмів машинного навчання для автоматизації виявлення дефектів, генерації тестових сценаріїв та оптимізації тестових наборів. Зокрема, новаторські підходи, такі як тестування на основі моделей (Model-Based Testing) та генерація тестів за допомогою AI, вже виявляють свою ефективність в умовах швидко змінюючихся вимог до програмного забезпечення. Впровадження інтелектуальних технологій в тестування сприяє автоматизації рутинних завдань, прискоренню виявлення помилок та покращенню загальної ефективності розробки.

Вибір конкретного підходу до автоматизації тестування має враховувати вимоги проекту, типи тестів та контекст розробки. Розмаїття інструментів та підходів надає розробникам широкий спектр варіантів для вибору найбільш ефективного рішення.

1.4 Огляд існуючих засобів автоматизованого тестування

1.4.1 Інструменти автоматизованого тестування веб-додатків

Серед найпоширеніших інструментів для автоматизованого тестування веб-додатків є Selenium. Він дозволяє розробникам ефективно створювати та виконувати тести на веб-сторінках у різних браузерах.

Selenium – це потужний фреймворк для автоматизації веб-тестування, який дозволяє розробникам ефективно виконувати тести на веб-сайтах на різних мовах програмування, таких як Java, Python, C#, і інші [2]. Функціональність Selenium включає в себе маневреність по веб-елементах, введення тексту, клікання, та перевірку стану веб-сторінок. Величезна спільнота користувачів та активна розробка роблять його одним із найбільш популярних інструментів для веб-автоматизації [3].

Selenium складається з трьох основних компонентів:

- Selenium WebDriver - це API, яке надає можливість взаємодії з веб-браузерами. WebDriver надає розширений інтерфейс для написання тестів та керування браузерами.
- Selenium Grid дозволяє виконувати тести паралельно на різних машинах та браузерах, що полегшує швидкість та ефективність тестування.
- Selenium IDE - розширення для браузера, яке дозволяє записувати та відтворювати дії користувача для швидкого створення тестових сценаріїв.

Selenium підтримує багато браузерів, таких як Chrome, Firefox, Safari, Edge, і Opera, що робить його ідеальним вибором для тестування крос-платформених додатків. Завдяки підтримці різних мов програмування, розробники можуть вибрати ту мову, з якою вони найбільш знайомі, для написання автоматизованих тестів. Selenium може бути легко розширений за допомогою різних плагінів і фреймворків, що дозволяє вам адаптувати його до конкретних потреб вашого проекту.

Іноді підтримка та утримання скриптів у Selenium може виявитися витратною задачею, особливо у великих проектах. Для цього важливо дотримуватися кращих практик програмування та використовувати паттерни проектування. У деяких випадках тести можуть виявитися непостійними через фрагільність або зміну елементів веб-сторінки. Ретельне проектування тестів та використання ідентифікаторів елементів може допомогти уникнути цих проблем.

Selenium є одним з найпопулярніших інструментів для автоматизованого тестування веб-додатків завдяки своїм функціональним можливостям та гнучкості. Використання Selenium у поєднанні з іншими інструментами, такими як фреймворки тестування та системи збірки, дозволяє розробникам створювати надійні, ефективні та повторювані тестові сценарії для забезпечення якості програмного забезпечення.

1.4.2 Фреймворки для написання тестів

Автоматизоване тестування включає використання спеціальних фреймворків, таких як JUnit чи TestNG для організації тестових сценаріїв, створення наборів тестів та автоматизованого виконання тестів.

JUnit є одним з найпопулярніших фреймворків для тестування за допомогою мови програмування Java [8]. Він надає структуру для написання тестових сценаріїв та виконання їх в автоматизованому режимі. JUnit спрощує процес виявлення помилок у програмному забезпеченні та забезпечує повторюваність результатів тестування. Використовуючи анотації, розробник може позначити методи як тести, налаштовувати перед тестуванням чи після, визначати групи тестів, тощо. JUnit може запускати тести в режимі групи чи окремо, дозволяючи розробнику визначати, які тести виконувати.

Однією з ключових переваг є простота структури тестових класів та методів. JUnit визначає стандартні правила для тестових умов, що полегшує їх створення та використання. Окрім базового функціоналу, JUnit надає додаткові можливості, такі як параметризовані тести, де один тест може бути викликаний з різними параметрами, що робить його більш гнучким та потужним. Ще однією значущою особливістю JUnit є його розширюваність. З можливістю використання різноманітних розширень і плагінів, розробники можуть налаштовувати фреймворк під свої потреби та легко інтегрувати його з іншими інструментами розробки та засобами автоматизації.

Загалом, JUnit є потужним, гнучким та добре підтримуваним фреймворком, який сприяє розробці високоякісного програмного забезпечення шляхом ефективного та автоматизованого тестування.

TestNG (Test Next Generation) - це фреймворк для тестування у середовищі Java, який надає розширені можливості порівняно з JUnit та став стандартом в багатьох сучасних Java-проектах. Вибір TestNG може бути обґрунтований більшою гнучкістю та розширеними функціональними можливостями [7].

TestNG використовує широкий спектр анотацій для визначення тестових методів, налаштування перед та після тестування, групування тестів. Можливість групувати тести дає змогу запускати тести з різних груп, контролювати їх порядок виконання, та використовувати групи для визначення тестових завдань. TestNG

дозволяє передавати параметри в тестові методи, що робить його потужним для випадків, коли тестовий сценарій повторюється з різними вхідними даними. Перевагами TestNG є можливість паралельного виконання тестів та розподілу навантаження, а також підтримка групування та налаштувань тестів, що забезпечує більшу гнучкість. Окрім цього, TestNG підтримує паралельне виконання тестів, що дозволяє значно зменшити час виконання тестового набору, розділяючи його на різні потоки або процеси.

Цей фреймворк може бути використаний для створення та виконання тестових сценаріїв, які включають в себе взаємодію з Selenium для автоматизованого тестування веб-додатків. Використання функцій TestNG, таких як групування тестів, параметризація, та залежності, може полегшити структуру та організацію тестового коду. Окрім того, можна використовувати HTML-звіти TestNG для візуалізації результатів та аналізу виконання тестів.

1.4.3 Огляд середовища розробки (IntelliJ IDEA)

Середовище розробки є ключовим інструментом для програмістів, яке значно впливає на продуктивність та якість програмного забезпечення. У магістерській роботі використовується IntelliJ IDEA – інтегроване середовище розробки, яке надає широкий набір інструментів для розробки та тестування програм [5]. IntelliJ IDEA має потужний редактор коду з різними функціями, такими як автодоповнення коду, рефакторинг, перегляд документації, інтеграція з системами контролю версій та підтримка багатьох мов програмування, включаючи Java, Kotlin, JavaScript, і інші.

IntelliJ IDEA підтримує широкий спектр плагінів, які дозволяють розширювати функціональність середовища розробки. Це може бути корисно для інтеграції з іншими інструментами, розширення підтримки мов програмування чи забезпечення нових можливостей. Також IntelliJ IDEA має вбудований дебагер, який дозволяє вам відстежувати виконання програми, встановлювати точки зупинки, аналізувати змінні, та використовувати інші інструменти для виправлення помилок.

Середовище розробки підтримує інтеграцію з системами управління залежностями, такими як Maven або Gradle. Це полегшує використання зовнішніх бібліотек та організацію структури проекту. IDEA надає різні інструменти для рефакторингу коду, такі як вилучення методу, переміщення класів, оптимізація імпортів, що сприяє поліпшенню якості коду та зменшенню ймовірності помилок. Окрім цього, IntelliJ IDEA забезпечує вбудовані інструменти для аналізу коду, виявлення помилок та попереджень, що сприяє поліпшенню якості програмного забезпечення та допомагає виявляти потенційні проблеми [6].

IntelliJ IDEA - потужне середовище розробки, яке відповідає вимогам сучасних проектів. Його інтегровані функції, зручний інтерфейс та велика кількість плагінів роблять його ефективним інструментом для розробників, зокрема у контексті магістерської роботи, де важлива автоматизація та тестування веб-додатків. Інтуїтивний інтерфейс IntelliJ IDEA, розширені функціональні можливості, а також можливість швидкої інтеграції зі сторонніми інструментами роблять його зручним для розробки та тестування. Інтеграція з Selenium, TestNG, та іншими інструментами тестування забезпечує легкість автоматизованого тестування веб-додатків.

1.4.4 Системи управління залежностями

Системи управління залежностями (Dependency Management Systems) відіграють важливу роль у розробці програмного забезпечення, дозволяючи ефективно керувати залежностями між різними компонентами проекту. У магістерській роботі використовуються інструменти для керування залежностями, такі як Maven або Gradle.

Maven - це потужний інструмент для автоматизації управління проектами та залежностями в середовищі Java. Maven використовує конфігураційні файли у форматі XML (POM) для опису проекту та його залежностей. Він забезпечує стандартизований підхід до створення проектів, збірки, публікації та управління бібліотеками.

Gradle - це система управління залежностями та збірка проектів, яка намагається поєднати переваги Maven і Apache Ant. Gradle використовує мову Groovy або Kotlin для опису конфігурації проекту та забезпечення гнучкості та простоти використання. Gradle використовує скрипти на основі мов Groovy або Kotlin для конфігурації проекту [4]. Ці скрипти є легкими та легко розширюваними. Ця система управління залежностями орієнтована на задачі. Кожен аспект збірки проекту представлений як задача, а плагіни розширюють його функціональність. Головною особливістю Gradle є декларативний підхід до конфігурації. Замість писання багатьох XML-тегів, ви визначаєте лише те, що вам потрібно. Перевагами є гнучка конфігурація та простота використання завдяки мовам Groovy та Kotlin, а також потужний механізм кешування та інкрементальної збірки для прискорення процесу збірки. Ще одною перевагою є здатність використовувати зовнішні залежності з різних репозиторіїв.

Обираючи між Maven та Gradle, слід враховувати специфіку проекту та власні уподобання. Maven надає стандартізований підхід та широкий спектр вбудованих функцій. З іншого боку, Gradle надає гнучку конфігурацію та легкість використання завдяки декларативній конфігурації та використанню мови програмування для опису проекту. Використання Gradle у магістерській роботі є обґрунтованим вибором, оскільки ця система управління залежностями пропонує швидку інкрементальну збірку та велику кількість плагінів для розширення функціональності проекту. Її інтеграція з іншими інструментами, такими як середовище розробки IntelliJ IDEA, забезпечує зручний та ефективний процес розробки та тестування.

1.4.5 Система Continuous Integration: Jenkins

Jenkins - це відкрите програмне забезпечення для неперервної інтеграції та доставки (CI/CD) в розробці програмного забезпечення. У магістерській роботі Jenkins використовується для автоматизації процесів тестування та розгортання, що забезпечує швидкий та надійний цикл розробки. Система діє як сервер неперервної інтеграції, який автоматизує процеси компіляції, тестування та розгортання коду з кожним зміненням в системі керування версіями.

Jenkins інтегрується з різними інструментами розробки, такими як системи контролю версій (Git, SVN), системи управління залежностями (Maven, Gradle), інструменти тестування (JUnit, TestNG), що дозволяє автоматизувати весь цикл розробки. Система працює на основі задач, які визначають конкретні дії, які повинні виконуватися. Плагіни розширюють функціональність Jenkins, надаючи можливості для виконання різноманітних завдань. Основною ідеєю Jenkins є

автоматична побудова проекту при зміні коду в системі контролю версій. Це може бути ініційовано вручну або автоматично.

Система надає зручний інтерфейс для моніторингу статусу збірок та результатів тестування. Також доступні детальні звіти, що полегшують аналіз помилок та вдосконалення процесу розробки. Jenkins забезпечує автоматизацію ключових етапів розробки, включаючи компіляцію, тестування та розгортання, що робить розробку ефективнішою та надійною.

Jenkins є універсальним інструментом, який підтримує багато мов програмування та технологій, що полегшує інтеграцію з різноманітними проектами. Велика кількість доступних плагінів дозволяє розширити функціональність для конкретних потреб проекту.

Jenkins є незамінним інструментом для неперервної інтеграції та доставки, що допомагає автоматизувати рутинні завдання розробки, забезпечуючи ефективність та надійність в робочих процесах. Використання Jenkins дозволяє створювати інтегровані та автоматизовані процеси тестування та розгортання, покращуючи якість програмного забезпечення.

1.5 Висновки до першого розділу

Обрані інструменти Selenium та Jenkins, разом із системою керування залежностями Gradle, свідчать про практичність та високий ступінь інтеграції в процес розробки. Інтеграція з IntelliJ IDEA підкреслює важливість комфортної розробки. Огляд стратегій тестування та систем керування процесами розкриває можливості ефективної імплементації автоматизованого тестування.

2 ПРОЕКТУВАННЯ ПРОГРАМНОЇ СИСТЕМИ

2.1 Специфікація вимог системи

2.1.1 Функціональні вимоги

а) Автоматизоване тестування веб-додатків з використанням Selenium

Система повинна надавати можливість розробки та виконання автоматизованих тестів для веб-додатків з використанням інструменту Selenium, а також мати можливість відтворення тестових сценаріїв для ефективного тестування різних функціональних частин веб-додатків.

б) Інтеграція з Jenkins для Continuous Integration

Система повинна підтримувати автоматичне виконання тестів під час кожного нового коміту або іншого події, що тригерить процес Continuous Integration. Забезпечення можливості налаштування та моніторингу запуску тестових сценаріїв через інтерфейс Jenkins.

в) Управління тестовими наборами

Система повинна дозволяти створення, редагування та виконання тестових наборів для організації та оптимізації тестових процесів. Окрім цього, повинна бута можливість групування тестів за функціональністю, пріоритетом або іншими параметрами для зручного виконання вибіркових тестових сценаріїв.

г) Генерація звітів та аналіз результатів

Система повинна автоматично генерувати детальні звіти після завершення тестування, включаючи інформацію про пройдені та невдалі тести, час виконання та інші метрики, а також забезпечувати можливість аналізу та порівняння результатів тестування для виявлення та виправлення помилок.

д) Підтримка різних веб-браузерів

Забезпечення можливості виконання тестів на різних веб-браузерах, таких як Chrome, Firefox, Safari тощо.

2.1.2 Нефункціональні вимоги

а) Ефективність та продуктивність

Програмна система повинна забезпечувати ефективне виконання тестів, інтегруючись з інструментами Selenium та Jenkins, і мінімізувати час, необхідний для завершення тестових сценаріїв. Також система повинна оптимізовано використовувати ресурси сервера та інших компонентів системи для запуску тестів, щоб уникнути перевантаження та забезпечити стабільну продуктивність.

б) Надійність

Справна робота системи є важливою, вона повинна бути стійкою та відновлюватися після непередбачених ситуацій, таких як збої апаратної чи програмної частини.

2.1.3 Вимоги до тестування

а) Тестові сценарії

Передбачається розробка тестових сценаріїв для функціональних та нефункціональних вимоги, щоб впевнитися в повноті тестового покриття. Також

визначення позитивних сценаріїв для перевірки правильності реалізації функцій та негативних тестів для виявлення можливих помилок та врахування виняткових ситуацій.

б) Параметризовані тестові сценарії

Вимагається підтримка параметризованих тестових сценаріїв для перевірки різних комбінацій вхідних даних та сценаріїв використання.

в) Тестування продуктивності

Тестування продуктивності вимагає визначення критерії та метрик ефективності системи, таких як час відповіді, швидкодія виконання тестів та інші. Вимагається визначення можливості масштабування системи для роботи з різною кількістю тестових ресурсів та паралельного виконання тестів.

г) Тестування сумісності

Для виявлення потенційних помилок системою передбачається сумічність із різними браузерами. Тому система підтримуватиме можливість виконання тестів на різних веб-браузерах та розробку механізмів автоматичної перевірки сумісності.

д) Тестування в умовах високого навантаження

Задля визначення меж можливостей системи та її оптимальної продуктивності, потрібне проведення тестів в умовах високого навантаження. Вимагається вивчення та тестування механізмів резервного копіювання та відновлення системи після виникнення відмов.

2.2 Архітектурне проектування системи автоматизованого тестування

Архітектурне проектування системи автоматизованого тестування визначає структуру та взаємодію компонентів, необхідних для реалізації функціональних та нефункціональних вимог. Проектування системи включає в себе вибір архітектурного стилю, організацію компонентів, визначення інтерфейсів та забезпечення масштабованості та легкості обслуговування. Архітектурне проектування системи повинно враховувати потреби у розширюваності, ефективності та зручності взаємодії з іншими інструментами та сервісами. Це забезпечує стабільність та надійність процесу автоматизованого тестування в умовах розвинутого розробницького середовища.

2.2.1 Вибір архітектурного стилю

Вибір архітектурного стилю для системи автоматизованого тестування є критичним етапом проектування, оскільки це визначає загальну структуру системи та взаємодію між її компонентами. Мета полягає в обранні такого стилю, який забезпечить ефективну, масштабовану та легко збережену систему.

Модульна архітектура визначається розділенням системи на окремі модулі, кожен із яких відповідає за конкретну функціональність. Це дозволяє створювати незалежні та легко розширювані компоненти, що полегшує підтримку системи та забезпечує її гнучкість.

2.2.2 Використання Page Object паттерну в системі автоматизованого тестування

Page Object паттерн є ефективним методом для структуризації автоматизованих тестів та підвищення їхньої читабельності, модульності та стабільності.

Основними принципами паттерну Page Object є розділення відомостей про сторінку. Кожна сторінка або фрагмент веб-додатку відображається в окремому класі Page Object, що містить всі відомості та дії, пов'язані з цією сторінкою. Ще однією особливістю цього паттерну є відокремлення тестових сценаріїв від деталей реалізації. Тестовий сценарій використовує методи Page Object для взаємодії з елементами сторінки, не вдаючись в деталі реалізації сторінки. Ізольовані Page Object класи можуть бути використані в різних тестових сценаріях, що забезпечує ефективне повторне використання коду.

Структура Page Object класу включає визначення елементів сторінки, методів для їх взаємодії та перевірок стану. Така організація дозволяє зробити тестовий код більш зрозумілим та стійким до змін в інтерфейсі додатку. Page Object класи можуть бути організовані відповідно до структури пакетів, що полегшує управління та підтримку. Це також сприяє покращенню масштабованості системи тестування, дозволяючи легко додавати нові Page Object класи при розширенні функціональності або додаванні нових сторінок.

У підсумку, використання Page Object паттерну при розробці системи автоматизованого тестування сприяє покращенню якості та надійності тестів, зменшенню затрат на їхню підтримку та створює підґрунтя для ефективної автоматизації тестового процесу.

2.2.3 Управління конфігурацією та залежностями

Управління конфігурацією та залежностями є важливою складовою будь-якого проекту, в тому числі і системи автоматизованого тестування. Використання інструменту Gradle для цих цілей є раціональним вибором, оскільки він забезпечує ефективний та гнучкий підхід до управління залежностями, конфігурацією та виконанням завдань.

Gradle використовує скрипти на базі мови програмування Groovy або Kotlin для визначення налаштувань проекту. Це дозволяє розробникам зручно визначати всі аспекти конфігурації та взаємодії залежностей в одному місці.

Файл build.gradle є ключовим файлом конфігурації проекту в системі Gradle. У випадку автоматизованого тестування з використанням бібліотек TestNG та Selenium, а також інших інструментів, цей файл визначає залежності, плагіни та інші налаштування проекту.

```
plugins {
    id 'java'
    id "io.freefair.lombok" version "8.4"
}

group = 'org.example'
version = '1.0-SNAPSHOT'

repositories {
    mavenCentral()
}

dependencies {
    testImplementation 'org.testng:testng:7.8.0'
    testImplementation 'org.seleniumhq.selenium:selenium-java:4.16.1'
    implementation 'com.aventstack:extentreports:5.0.9'
}

test {
    useTestNG()
}
```

Рисунок 2.1 – Підключення бібліотек та плагінів у файлі build.gradle

На рисунку 2.1 зображено використання функціоналу Gradle:

- plugins: визначає плагіни, які використовуються в проекті. У цьому випадку, java плагін вказує, що проект є проектом Java. Плагін Lombok дозволяє спростити написання коду за допомогою аннотацій, таких як Data, Getter, Setter. Це допомагає уникнути написання багато однотипного коду для генерації методів доступу та інших стандартних операцій.
- repositories: вказує репозиторії Maven, з яких будуть завантажуватись залежності.
- dependencies: визначає залежності проекту. В даному випадку, визначені бібліотеки для Selenium, TestNG та ExtentReport.
- test: конфігурація для тестів. useTestNG() вказує, що проект використовує TestNG для запуску тестів.

Файл build.gradle дозволяє легко встановлювати та керувати версіями бібліотек, а також налаштовувати тестове середовище для проекту з використанням Selenium, TestNG, ExtentReport та Lombok.

2.3 Проектування модулів системи

2.3.1 Функціональні можливості веб-додатку ars.ua

Для створення програмної системи автоматизованих тестів було обрано інтернет-магазин компанії ТОВ “Торгова компанія “АРС-Кераміка”.

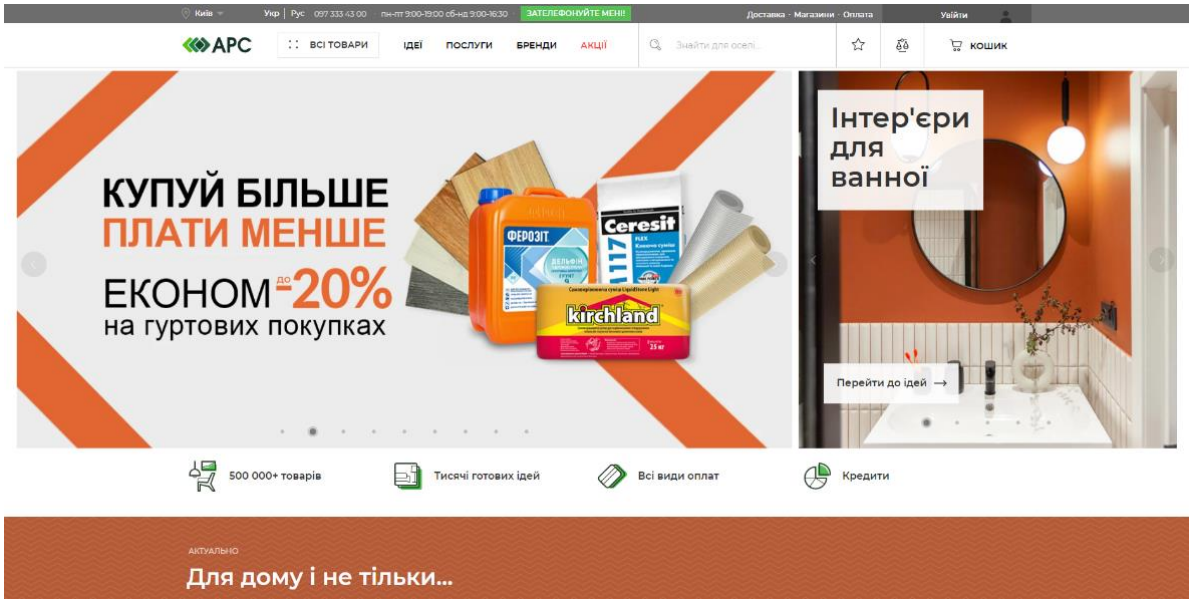

Рисунок 2.2 – Основна сторінка веб-додатка

Даний веб-додаток має ряд функціональних можливостей:

- відображення товарів або послуг, які пропонуються компанією, у зручному для користувача форматі;
- можливість знаходження товарів за допомогою пошуку та фільтрації за різними параметрами, такими як категорія, бренд, ціновий діапазон і т.д;
- реєстрація та авторизація користувачів, можливість переглядати та відстежувати їхні замовлення, зберігати улюблені товари, налаштовувати персональні налаштування;
- додавання товарів до кошика, перегляд та зміна його вмісту, оформлення замовлення;
- вибір методів оплати, введення адреси доставки, вибір служби доставки;
- форма для зворотного зв'язку, контактні дані;
- інформація про поточні акції, розпродажі та спеціальні пропозиції;

2.3.2 Створення модулів взаємодії з веб-сторінкою

Важливим етапом є визначення стратегії взаємодії з елементами веб-сторінок. Розглядається вибір оптимальних стратегій локаторів для ефективного знаходження та взаємодії з елементами, такими як кнопки, поля вводу, чекбокси та інші.

Окрім цього, розробляються модулі для роботи з динамічними елементами сторінок та обробки асинхронних подій. Вивчається використання засобів для очікування та синхронізації, щоб забезпечити стабільність взаємодії при завантаженні сторінок. Ключовим аспектом є також розгляд рефакторингу та підтримки коду взаємодії з веб-сторінкою. Визначаються стандарти коду, які сприяють читабельності, модульності та підтримці тестового коду в довгостроковій перспективі.

Перш за все, потрібно визначити стратегію розробки Page Object моделі для кожної унікальної веб-сторінки.

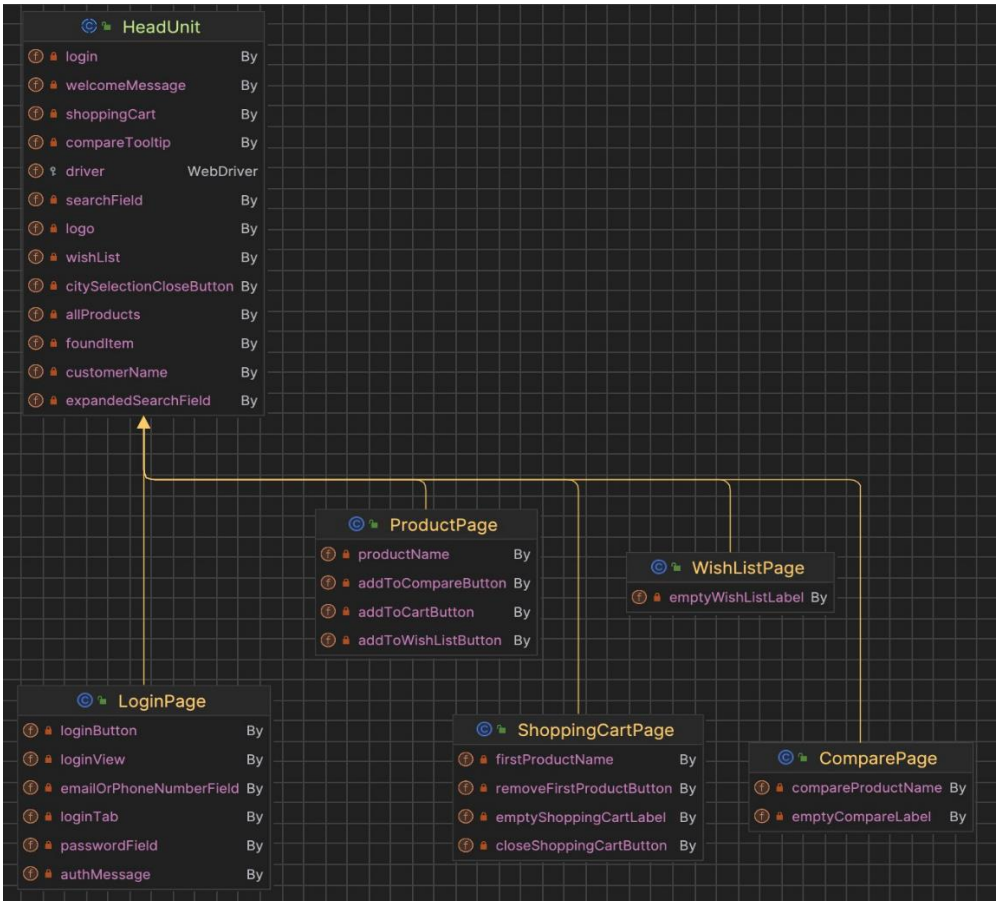

Рисунок 2.3 – UML-діаграма класів в пакеті pages

На рисунку 2.3 зображено ієрархію класів в пакеті pages. Усі класи наслідуються від HeadUnit, оскільки цей елемент веб-сторінки є спільним для усіх інших сторінок. Дані класи містять локатори для пошуку елементів відповідних сторінок, а також методи для взаємодії з цими елементами (натискання на кнопку, отримання тексту елементу, введення тексту в поле).

Також класи містять функціональні методи, які в подальшому використовуються у бізнес-логіці та автоматизованих тестах.

```java
// Functional
protected void fillInLoginForm(String email, String password) {
    openLoginTab();
    enterEmail(email);
    enterPassword(password);
    clickCompleteLogin();
}

// Business Logic
public HomePage successfulLogin(String email, String password) {
    fillInLoginForm(email, password);
    return new HomePage(driver);
}

public LoginPage unsuccessfulLogin(String email, String password) {
    fillInLoginForm(email, password);
    return new LoginPage(driver);
}
```

Рисунок 2.4 – Методи бізнес-логіки в класі LoginPage

2.3.3 Модуль забезпечення тестової конфігурації

Цей модуль включає в себе інструменти та методи, які дозволяють налаштовувати параметри виконання тестів, забезпечуючи гнучкість та ефективність в різних сценаріях тестування.

Рисунок 2.4 – UML-діаграма класу TestRunner

Клас TestRunner відповідає за налаштування тестів за допомогою анотацій фреймворку TestNG. У цьому класі вказані конфігураційні параметри для тестів, такі як браузер, URL та інші, які будуть використовуватись в тестових методах.

Метод beforeClass() здійснює налаштування веб-драйвера, а також веб-браузера. Цей метод позначений анотацією @BeforeClass, тому він буде буде викликаний один раз перед виконанням будь-яких тестів у класі. Це важливо, оскільки цей метод містить логіку, яка повинна бути виконана лише один раз перед запуском тестів усього класу.

Метод afterClass() використовує анотацію @AfterClass та призначений для закриття веб-браузера та з’єднань з веб-драйвером після виконання усіх тестів. Анотація @AfterClass використовується для виконання завершальних дій після виконання всіх тестів у класі.

Метод setUp() має анотацію @BeforeMethod і викликається перед кожним тестовим методом у класі. Це дозволяє виконати певні дії перед кожним тестом, а не тільки один раз для всього класу тестів. Цей метод виконує відкриття веб-

сторінки на якій будуть виконуватись автоматизовані тести, а також очікує на повне її завантаження.

Метод tearDown() позначений анотацією @AfterMethod, буде викликаний після виконання кожного тестового методу у класі. Це дозволяє виконати завершальні дії після кожного окремого тесту. Цей метод виконує перевірку результату виконання тесту і у випадку помилки, зберігає знімок екрана. Окрім цього, tearDown() відповідальний за очищення файлів cookie після кожного тесту, а також зберігає всі дані для ExtentReport звіту.

2.3.4 Допоміжні модулі

Модуль для забезпечення звітності є важливою частиною системи автоматизованого тестування і відповідає за створення докладних та інформативних звітів після виконання тестів. У цьому контексті, модуль використовує бібліотеку ExtentReport для генерації зрозумілих звітів.

Рисунок 2.5 – UML-діаграма класу ExtentReprotManager

Модуль відповідає за збір ключової інформації під час виконання кожного тесту. Ця інформація може включати результати тесту, час виконання, статус (пройшов/не пройшов), а також будь-які зіткнення чи винятки, що виникли.

Використовуючи ExtentReport, модуль створює докладні звіти, які мають чітку структуру та графічні елементи. Ці звіти можуть містити відомості про кожен тест, його вхідні дані, очікувані та фактичні результати. Даний модуль забезпечує візуалізацію результатів тестів у вигляді графіків, діаграм та інших графічних елементів, що роблять звіти більш зрозумілими та легкими для аналізу.

Модуль може бути налаштований для інтеграції з системами Continuous Integration/Continuous Deployment (CI/CD), щоб автоматично створювати та оновлювати звіти при кожному запуску тестового набору.

Модуль StringConstants є частиною системи автоматизованого тестування і відповідає за зберігання та управління константами, які використовуються в різних частинах коду проекту. Використання такого модулю допомагає уникнути магічних чисел та рядків у коді, забезпечуючи централізоване місце для зберігання всіх текстових констант та покращуючи читабельність коду.

Допоміжний модуль TestUtils є важливою частиною системи автоматизованого тестування і включає набір функцій, призначених для полегшення написання тестового коду та забезпечення його ефективності та стабільності. Цей модуль призначений для забезпечення зручних та ефективних методів взаємодії з користувацьким інтерфейсом (UI) в процесі написання автоматизованих тестів. Модуль містить методи для взаємодії з різними типами веб-елементів, що дозволяє легко виконувати дії з елементами UI. TestUtils має функції для очікування певного стану елементів, наприклад, чи вони видимі, активні чи доступні для взаємодії.

Використання готових методів для очікування та взаємодії з елементами допомагає уникати помилок та збільшити стабільність автоматизованих тестів.

Також модуль допомагає зробити взаємодію з елементами користувацького інтерфейсу більш зручною та зрозумілою.

Рисунок 2.6 – UML-діаграма класу TestUtils

2.3.5 Модуль реалізації функціонального тестування

Smoke тестування є типом тестування, який спрямований на перевірку базової функціональності програмного забезпечення після його встановлення чи оновлення. Це першочерговий, швидкий та обмежений за обсягом вид тестування, призначений для виявлення серйозних помилок або проблем, які можуть призвести до відмови в роботі системи.

Основна мета Smoke тестування - забезпечити впевненість у тому, що основні функції програмного забезпечення працюють коректно після важливих змін чи оновлень. Це включає в себе перевірку запуску програми, можливості авторизації користувача, виконання ключових функцій та взаємодії з базовими компонентами системи.

Smoke тестування є ефективним інструментом для раннього виявлення серйозних проблем та забезпечення базової стабільності системи перед більш детальними етапами тестування. Використання цього виду тестування дозволяє оптимізувати витрати та прискорює процес визначення готовності системи до подальших тестів.

Рисунок 2.7 – UML-діаграма класу SmokeTests

Клас SmokeTests визначає набір тестових сценаріїв та наслідується від класу TestRunner. Його мета - перевірка базової функціональності системи та виявлення серйозних проблем або відхилень від очікуваного стану. Цей клас є важливим інструментом для раннього виявлення серйозних проблем та забезпечення базової стабільності системи.

2.4 Інтеграція з Jenkins для забезпечення Continuous Integration

Налаштування Jenkins включає в себе кілька ключових кроків для забезпечення ефективності та надійності процесу Continuous Integration (CI) [9]. Після встановлення Jenkins і запуску його веб-інтерфейсу, введення пароля адміністратора та створення облікового запису, важливо налаштувати конфігурацію проекту. Це включає вибір типу проекту (Freestyle або Pipeline), налаштування параметрів, таких як джерело коду та скрипти збірки [10].

Плагіни в Jenkins є невід'ємною частиною екосистеми системи Continuous Integration. Ці розширення додають різноманітні можливості, роблячи Jenkins більш гнучким та пристосованим до конкретних потреб розробки. Завдяки плагінам, можна інтегрувати різні інструменти, автоматизувати процеси, та розширювати можливості системи.

Створення та запуск проекту включає в себе обрання типу збірки та визначення необхідних кроків, таких як встановлення залежностей, виконання тестів та збірка програми. Постійна інтеграція реалізується через регулярні автоматизовані збірки, які автоматично запускаються при виявленні змін у кодовій базі.

Основною метою Jenkins є створення прозорого та ефективного процесу CI, який дозволяє вчасно виявляти та виправляти помилки, забезпечує надійність збірок та поліпшує спільну роботу розробницької команди.

Створення та налаштування Jenkins Pipeline - це процес визначення послідовності дій для автоматизації CI/CD процесів.

Конфігурація Jenkins Pipeline передбачає визначення параметрів, таких як вибір тестового фреймворку, налаштування залежностей та використання спеціалізованих звітів (наприклад ExtentReport) для аналізу результатів тестування. Метою є автоматизація та оптимізація виявлення та виправлення помилок в коді під час розробки.

Створюючи Pipeline Script, варто приділити увагу визначенню тестових етапів, включаючи підготовку тестового середовища, виконання тестових сценаріїв та аналіз результатів.

```
pipeline {
    agent any

    stages {
        stage('Checkout') {
            steps {
                // Checkout the code from Git repository
                script {
                    git branch: 'master', credentialsId: '{6ab600a5-7a71-4b43-a9e8-e45735fe6636}', url: 'https://Milnik@bitbucket.org/Milnik
                }
            }
        }

        stage('Build and Test') {
            steps {
                // Run Gradle build and tests
                script {
                    sh './gradlew clean test'
                }
            }
        }
    }
    post {
        always {
            script {
                publishHTML(target: [
                    allowMissing: false,
                    alwaysLinkToLastBuild: true,
                    keepAll: true,
                    reportDir: 'build/reports/tests/test/',
                    reportFiles: 'index.html',
                    reportName: 'Test Summary'
                ])

                publishHTML(target: [
                    allowMissing: false,
                    alwaysLinkToLastBuild: true,
                    keepAll: true,
                    reportDir: 'build/reports/extentReport/',
                    reportFiles: 'extent_report.html',
                    reportName: 'Extent Report'
                ])
            }
        }
    }
}
```

Рисунок 2.8 – Налаштування Pipeline Script

Даний скрипт, написаний на мові програмування Groovy, виконує викачування тестового коду з Git-репозиторію в першому етапі. В наступному етапі відбувається запуск і виконання тестових сценаріїв. Під час фінального етапу post

виконується публікація HTML звітів, згенерованих за допомогою фрейморків TestNG та ExtentReport.

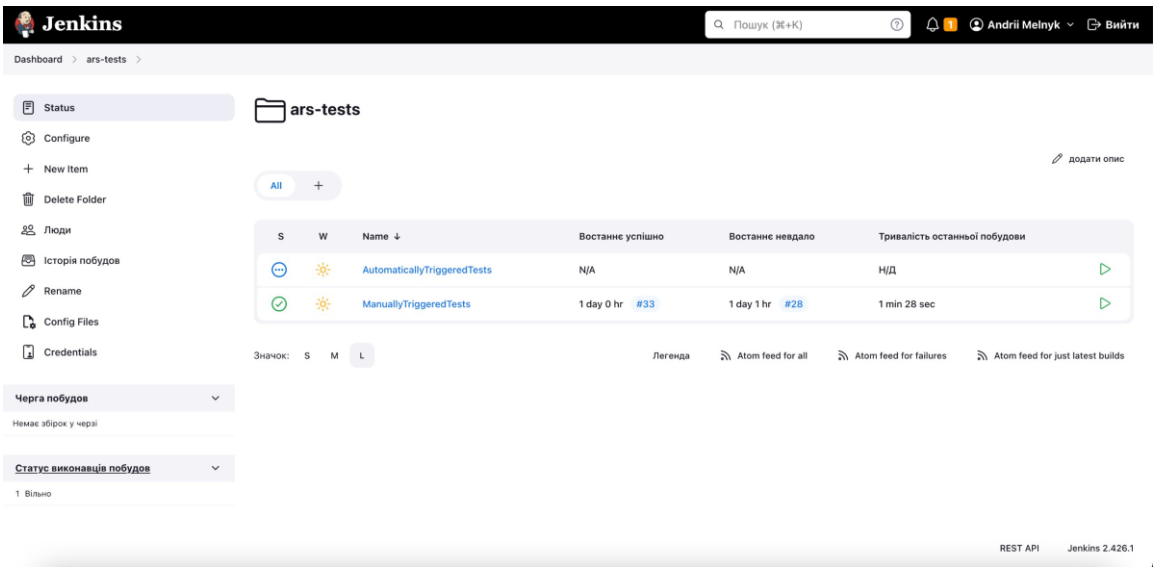

Рисунок 2.9 – Користувацький інтерфейс Jenkins та налаштовані Jenkins Pipeline для виконання тестових сценаріїв

– AutomaticallyTriggeredTests – збірка, яка запускатиметься автоматично за заданими критеріями(після додавання нових змін у тестовому коді або за заданим часовим проміжком).
– ManuallyTriggeredTests – збірка, яку можна запустити вручну.

2.5 Висновки до другого розділу

В даному розділі було ретельно розглянуто архітектурні та функціональні аспекти системи автоматизованого тестування. Описані модулі та компоненти системи, визначено їхні функціональні можливості та вибрані оптимальні інструменти для розробки. Запропоновані рішення враховують вимоги до ефективності,

масштабованості та підтримки легкої модифікації, що є ключовими аспектами подальшого успішного розвитку системи.

3 ТЕСТУВАННЯ РОЗРОБЛЕНОЇ ПРОГРАМНОЇ СИСТЕМИ. АНАЛІЗ ЇЇ РОБОТИ

3.1 Тестування за допомогою Gradle

Тестування за допомогою Gradle відіграє ключову роль у забезпеченні ефективності та надійності розробленої програмної системи. Gradle, як система управління залежностями та збірки проектів, надає широкий спектр можливостей для автоматизації тестового процесу. Використання Gradle дозволяє легко налаштовувати та виконувати тести, надаючи гнучкість у конфігурації тестових задач. Можливість інтеграції з іншими інструментами тестування, зокрема TestNG, робить Gradle потужним інструментом для запуску та керування різними типами тестів. Параметризація тестування в Gradle дозволяє створювати різні конфігурації для виконання тестових сценаріїв в різних умовах, що спрощує тестування на різних середовищах та конфігураціях.

Після успішного визначення налаштувань у файлі build.gradle (рисунок 2.1) та написання тестових сценаріїв у відповідних класах, можна переходити до етапу виконання тестів. Знаходячись у терміналі або командному рядку та перейшовши до кореневої папки проекту, потібно виконати команду gradle test. Цей крок дозволить впевнитися, що тести виконуються правильно та проходять усі необхідні перевірки. Окрім того, в Gradle є можливість передавати параметри, наприклад, -Pbrowser=chrome, для специфікації конкретного браузера для виконання тестів. Це робить процес тестування більш гнучким та адаптованим до конкретних умов.

3.2 Тестування за допомогою Jenkins

Jenkins є ключовим інструментом для автоматизації тестування та підтримки стабільності програмного продукту протягом усього циклу розробки. Налаштований проект включає в себе попередньо сконфігуровані налаштування, такі як параметри зберігання конфігурації, з'єднання з репозиторієм та завдання для збирання проекту. Тестовий пайплайн вже визначений, і його можна запускати для виконання тестів.

Запуск тестового пайплайну в Jenkins автоматично ініціює процес тестування, включаючи виконання автоматизованих тестів та генерацію звітів. Результати тестів та звіти доступні для перегляду в інтерфейсі Jenkins, що дозволяє швидко визначити стан якості програмної системи.

Завдяки встановленим налаштуванням та інтеграції з Jenkins, тестування стає ефективним та прозорим етапом розробки, забезпечуючи швидке виявлення та виправлення помилок, а також забезпечуючи неперервну якість програмного продукту.

3.3 Виявлення та обробка помилок

Виявлення помилок включає в себе систематичний аналіз вихідних даних, внутрішніх журналів та звітів про виконання тестів. Застосовуються різні методи, такі як контроль даних на вході, моніторинг ресурсів та використання інструментів реєстрації подій для виявлення неправильностей.

Після виявлення помилок важливо визначити їхні причини та можливі наслідки для системи. Обробка помилок включає в себе автоматичне відновлення роботи системи в стан, придатний до використання, або генерацію повідомлень для адміністраторів та кінцевих користувачів.

Окрім того, варто приділити увагу розробці та впровадженню стратегій резервного копіювання та відновлення для мінімізації можливих наслідків великих системних помилок. Всі ці заходи спрямовані на забезпечення найвищого рівня стабільності та доступності системи в умовах реальної експлуатації.

3.4 Результати тестування

Тестування виявилося успішним і продемонструвало ключові аспекти стабільності та працездатності системи. У першу чергу, система успішно пройшла базові перевірки, включаючи запуск головної функціональності та інтерфейсу користувача. Суттєвих помилок або несправностей в роботі виявлено не було.

Виконання автоматизованих тестів дозволило впевнитися у функціональності основних компонентів системи та їх коректній взаємодії. Важливо відзначити, що під час smoke тестування не виявлено критичних проблем, які могли б позначитися на загальній працездатності системи. Такий результат

свідчить про базову стабільність розробленої системи та готовність до більш глибокого та детального тестування.

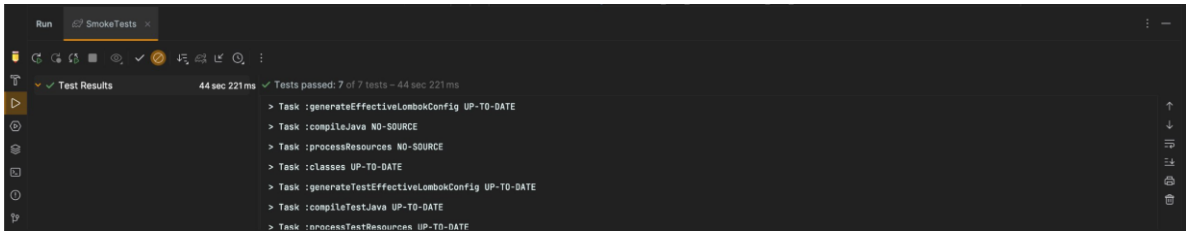

Рисунок 3.1 – Результат виконання автоматизованих тестів в IntelliJ IDEA

Тестування на Jenkins підтвердило коректну інтеграцію та працездатність системи. Було успішно перевірено базові аспекти, такі як запуск та завершення збірки проекту, налаштування з'єднання з репозиторієм, а також запуск тестів автоматизованого тестування через Jenkins.

Smoke тестування не виявило серйозних аномалій або суттєвих помилок в роботі Jenkins. Загальна стабільність системи та її здатність до базового функціоналу були підтверджені. Це дозволяє вважати Jenkins готовим до подальшого використання та інтеграції в процес CI/CD для автоматизованого тестування проекту.

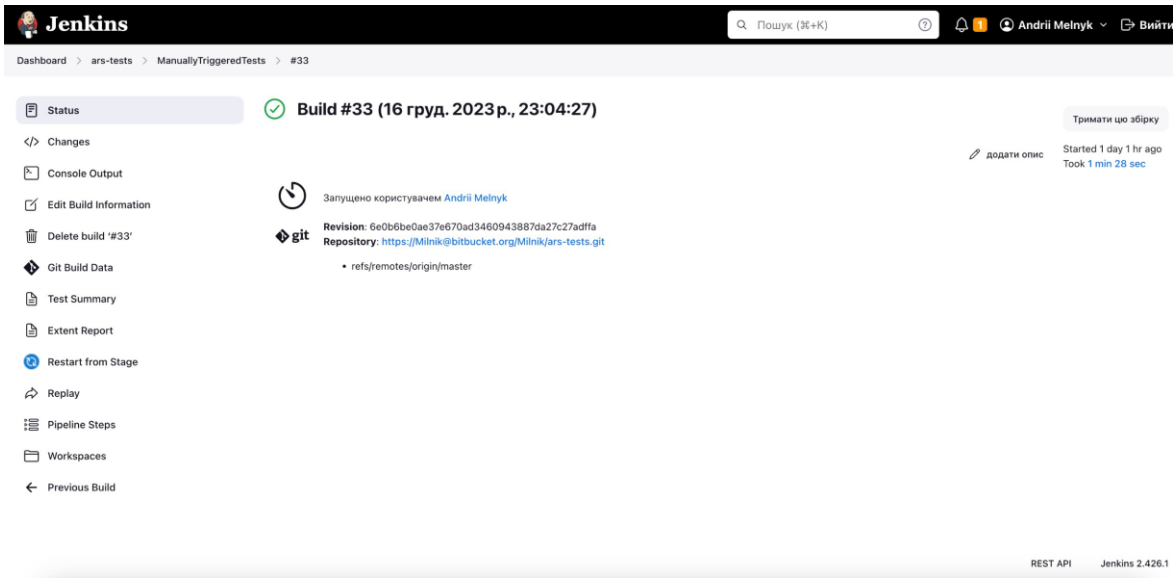

Рисунок 3.2 – Результат виконання автоматизованих тестів на Jenkins

Усі тести були виконані 5 разів, у жодному запуску не було виявлено помилок чи невдалих виконань тестових сценаріїв.

Функціональне тестування було спрямовано на перевірку відповідності функціональним вимогам, визначеним у попередніх етапах проекту. Проведений аналіз результатів тестування підтверджує високу ефективність розробленої системи автоматизованого тестування. Під час виконання тестових сценаріїв відзначено коректну реакцію системи на вхідні дані та відповідність її функціоналу визначеним вимогам. Аналіз стійкості системи до навантаження також підтвердив її готовність до роботи в реальних умовах. Отримані під час тестування результати дозволяють зробити висновки щодо готовності системи до подальших етапів впровадження та використання в реальних умовах.

3.5 Аналіз згенерованих звітів

Аналіз згенерованого звіту за допомогою TestNG вказує на ключові аспекти виконання автоматизованих тестів та надає важливу інформацію для оцінки якості та стабільності тестового набору.

У звіті визначається кількість виконаних тестів, їхній стан (успішні, невдалий чи пропущені), а також час, необхідний для виконання кожного тесту. Це дозволяє здійснити перший висновок про ефективність тестового набору та ідентифікувати конкретні тести, які можуть потребувати уваги.

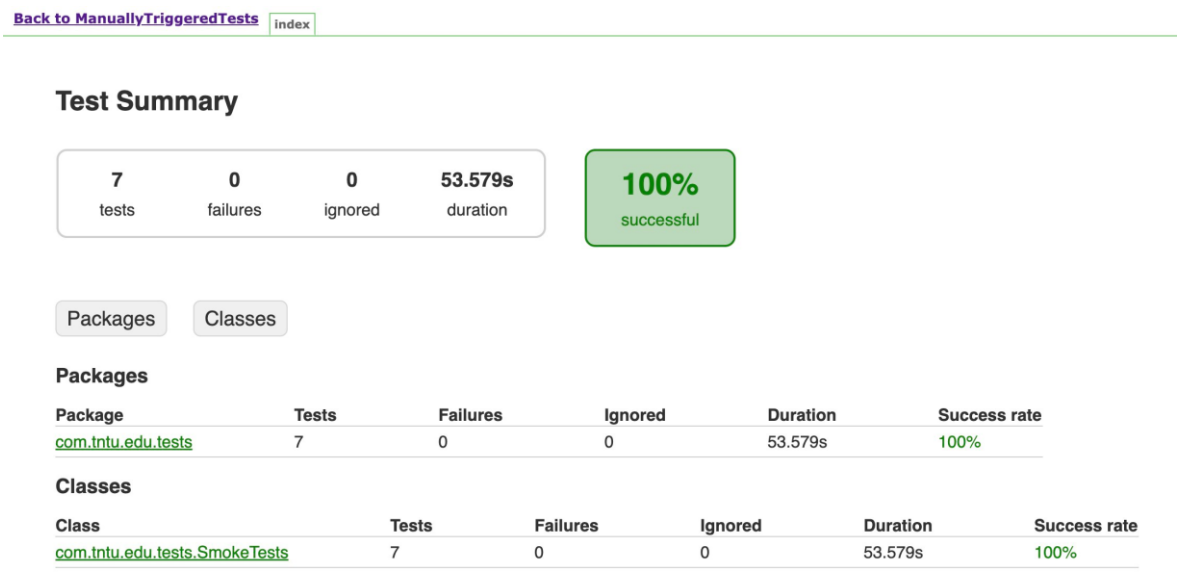

Рисунок 3.3 – Звіт про виконання автоматизованих тестів згенерованих за допомогою TestNG

Аналіз згенерованого звіту за допомогою ExtentReport виявився дуже інформативним і корисним. Загальна картина результатів тестування дозволяє швидко оцінити стан тестового набору та виявити проблеми.

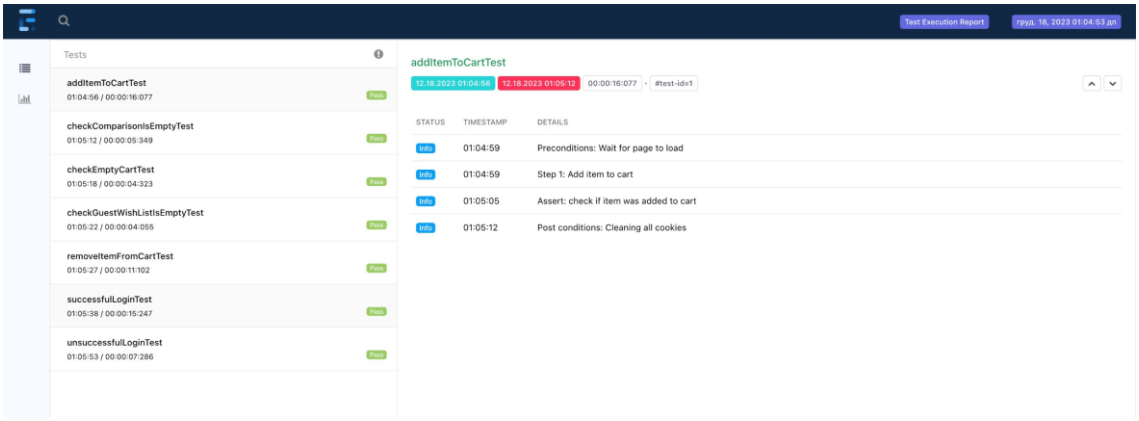

Рисунок 3.4 – Звіт про виконання автоматизованих тестів згенерованих за допомогою ExtentReprot

Деталізація кожного тесту у звіті дозволяє вивчати важливі аспекти, такі як стан виконання, час виконання та додаткові метрики. Логи та стек-трейси помилок полегшують виявлення та усунення дефектів. Можливість перегляду знімків

екрану з тестів дозволяє краще зрозуміти контекст помилок та спростити їхнє виправлення.

Графіки та інші графічні елементи звіту надають зрозуміле зображення ключових метрик тестування.

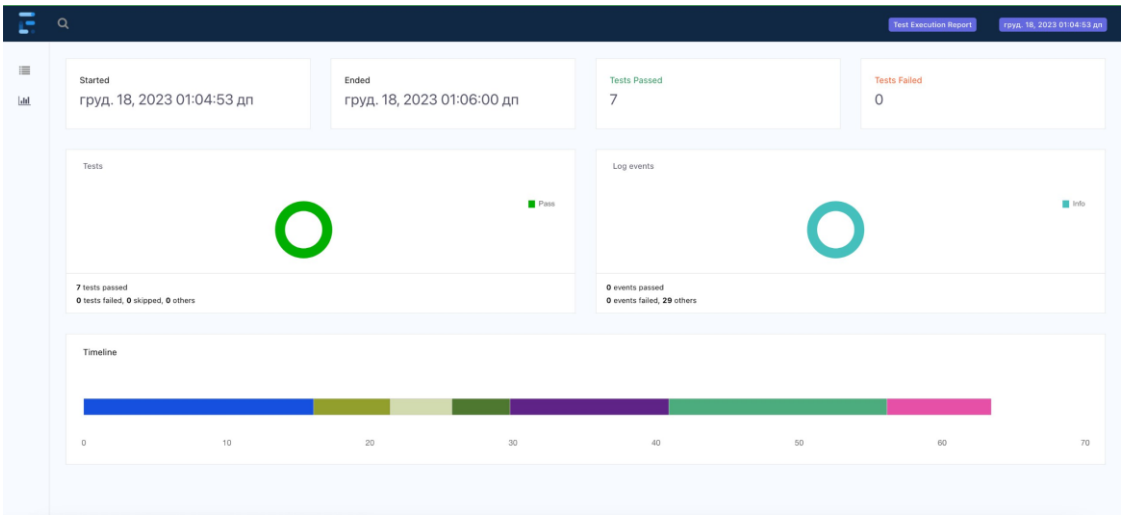

Рисунок 3.5 – Графіки та метрики виконаних автоматизованих тестів згенерованих за допомогою ExtentReprot

У випадку виявлення невдалого виконання тестового сценарію до звіту додається знімок екрану в момент помилки.

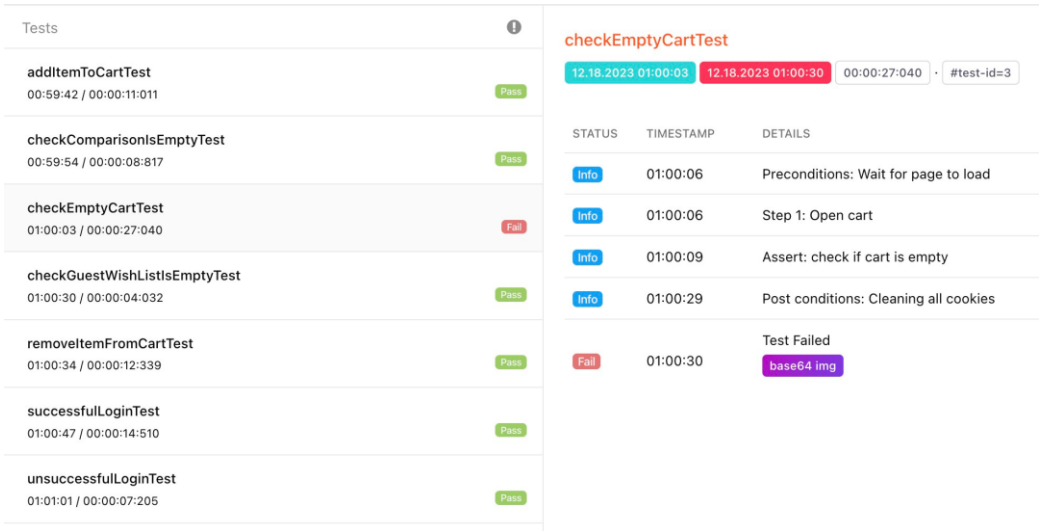

Рисунок 3.6 – Невдале виконання автоматизованого тесту

Загалом, ExtentReport надає зручний інструмент для аналізу результатів тестування та прийняття обґрунтованих рішень для подальшого удосконалення якості розроблюваної системи.

3.6 Аналіз використання та оптимізації ресурсів

Після успішного налаштування та виконання автоматизованих тестів у рамках розробленої системи, проведено аналіз використання ресурсів з метою їхньої оптимізації. У процесі виконання тестів було визначено, що обчислювальні ресурси використовуються ефективно, існує можливість оптимізації витрат оперативної пам'яті та управління нею для забезпечення стабільності тестового середовища.

Також була проведена оцінка мережевих ресурсів, що підтвердила їх ефективне використання під час взаємодії системи з різними компонентами та сервісами. Вивчений дисковий простір показав задовільне використання та ефективне збереження результатів тестів. Час виконання тестів також було проаналізовано, і виявлені шляхи його оптимізації.

Окремо було звернено увагу на раціональне використання ресурсів під час виконання тестів у Jenkins. За допомогою налаштованих пайплайнів вдалося досягти оптимального використання обчислювальних та мережевих ресурсів для забезпечення ефективності автоматизованого тестування.

Загалом, аналіз використання та оптимізація ресурсів підтвердив ефективність налаштованої системи автоматизованого тестування та ідентифікував можливості для подальшої оптимізації з метою підтримки стабільності та швидкодії в тестовому середовищі.

3.7 Висновки до третього розділу

Використання Page Object паттерну та модульного підходу до проектування тестів сприяло покращенню підтримуваності та розширюваності тестового набору. Інтеграція з Jenkins дозволила автоматизувати процес виконання тестів та забезпечила Continuous Integration.

Автоматизовані тести, використовуючи інструменти Selenium, дозволили швидко та ефективно перевірити функціональність системи та забезпечити високий рівень покриття коду. Аналіз звітів від TestNG та ExtentReport підкреслив важливість інформації, яку надає візуалізація результатів, логів та відомостей про виконання тестів.

Дані висновки свідчать про те, що використання автоматизованого тестування та сучасних інструментів сприяло покращенню якості програмного продукту, скороченню часу розробки та підвищенню рівня довіри до системи з боку користувачів.

4 ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ

4.1 Охорона праці

Проведені дослідження та розробка системи автоматизованого тестування відбувалося з використанням персональних комп’ютерів. Тому важливим фактором безпеки праці є дотримання основних правил користування комп’ютерною технікою, основних норм та правил охорони праці. Для цього потрібно забезпечити користувачам максимально комфортні та безпечні умови для їх перебування в приміщенні для того, щоб вони якісно та ефективно виконували поставлені завдання.

Охорона праці – це система правових, соціально-економічних, організаційно-технічних, санітарно-гігієнічних і лікувально-профілактичних заходів та засобів, спрямованих на збереження життя, здоров'я і працездатності людини у процесі трудової діяльності. Забезпечення відповідних умов праці повинні контролювати власник або уповноважений ним орган.

Варто звернути увагу на відповідність робочого процесу законодавчим та стандартним вимогам у галузі охорони праці, а також на важливість регулярного оновлення та підвищення кваліфікації з питань безпеки праці. Робоче місце користувача комп’ютером повинно відповідати ергономічним вимогам ДСТУ 8604:2015 Дизайн і ергономіка. Робоче місце для виконання робіт у положенні сидячи. Загальні ергономічні вимоги з урахуванням характеру та особливостей трудової діяльності. При розміщенні подібних робочих місць необхідно дотримуватись наступних вимог:

- відстань між бічними поверхнями, на яких розміщений обчислювальний прилад, не має бути меншою ніж 1,2 м;
- інтервал між рядами робочих місць має бути не менш 1 м.

Вимоги цих пунктів щодо відстаней та інтервалів беруться з урахуванням особливостей стін та перегородок.

Робоче місце користувача комп’ютером повинно забезпечувати збереження задовільної робочої пози за наступними рекомендованими правилами: ноги – на підлозі або на підставці для ніг, стегна – в горизонтальному положенні; монітор комп’ютера – не менше 60 см від обличчя користувача.

Клавіатуру необхідно розташовувати на поверхні столу або на спеціальній, регульованій за висотою, робочій площині окремо від столу на відстані 10-30 см від краю, ближчого до працівника.

Робочий стілець повинен бути оснащений наступними елементами: сидінням, спинкою та стаціонарними або змінними підлокітниками. Стілець може бути додатково обладнаний засобами, що поліпшують умови роботи, а саме: підголовник та підставка для ніг. Робочий стілець користувача комп’ютером повинен регулюватись за висотою, кутом нахилу сидіння та спинки, за відстанню спинки до переднього краю сидіння, висотою підлокітників. Ширина та глибина сидіння повинні бути не меншими за 40 см. Висота поверхні сидіння має регулюватися в межах 40-50 см,

Попередження втомленості та стресу є важливою складовою забезпечення ефективної та безпечної праці під час розробки та використання системи автоматизованого тестування. Для досягнення цієї мети, важливо враховувати принципи ергономіки робочого місця, включаючи оптимальне розташування моніторів, правильне освітлення, налаштування робочого часу, та періодів відпочинку. Також слід вдосконалювати методи взаємодії з комп'ютером, використовуючи ефективні інструменти та враховуючи психологічні аспекти навантаження. Проведення періодичних перерв у роботі, фізична активність та застосування технік відпочинку можуть сприяти зниженню рівня стресу та підтримувати оптимальний робочий режим. Увага до цих аспектів допомагає створювати здорове та продуктивне робоче середовище.

Таким чином, в рамках розробки системи враховані основні аспекти безпеки праці, таких як ергономіка робочого місця, попередження втомленості та стресу, використання безпечних методів роботи з комп’ютером та програмним забезпеченням. Аналіз та врахування аспектів охорони праці є необхідним етапом для створення сприятливого та безпечного робочого середовища для людини, що працює над розробкою та експлуатацією системи автоматизованого тестування.

4.2 Фактори ризику і можливості ураження здоров'я користувачів комп'ютерів

Близько 20% порушень здоров’я, пов’язаних із роботою за комп’ютером, викликано не дією шкідливих чинників, які генерує комп’ютер, а незнанням основних правил роботи з ним та неправильною організацією робочого місця.

Робота за комп’ютером є шкідливою, — це вперше зрозуміли в найбільш комп’ютеризованій країні світу — США. За даними Міністерства праці цієї країни, одна тільки «травмуюча дія, що повторюється при роботі з комп’ютером, коштує корпоративній Америці в 100 млрд доларів щорічно, а компенсації, виплачені службовцям, які працюють з комп’ютерами, досягають астрономічних розмірів.

Виділимо окремі аспекти негативного впливу на стан здоров’я користувачів персональних комп’ютерів та методи усунення такого впливу.

Сукупність змін, що спостерігаються в стані здоров’я користувачів ЕОМ, включає у себе захворювання органів зору, опорно-рухового апарату, центрально-нервової і серцево-судинної систем, шлунково-кишкового тракту, алергійні розлади. Виконання робіт із застосуванням комп’ютера, також може спричинити розвиток професійних захворювань периферичної нервової системи; захворювання кістково-м’язової системи та сполучної тканини, інші захворювання:

У вагітних відзначається ускладнення вагітності та пологів. У 80% працівників, які виконують роботу, що вимагає постійного напруження зору (а саме такою є робота на комп’ютері), відзначається прогресуюче зниження працездатності, що наступає через 45-60 хв після початку роботи і поступово призводить до перевтоми, розладів центральної нервової та інших систем організму. У другій половині дня (іноді раніше) з’являється втома, головний біль, біль в очах та інші скарги, а швидкість переробки інформації зменшується на 25-

34%, знижується стійкість ясного бачення на 40-52%. До кінця робочого дня збільшується частота серцевих скорочень і показники систолічного і діастолічного артеріального тиску.

Вже в перші роки комп'ютеризації було відзначено специфічне зорове стомлення у користувачів ЕОМ, яке дістало назву астенопія. За різними даними , частота випадків астенопії коливається від 40-92% (періодично) до 10-40% (щодня). Деякі симптоми астенопії можуть зберігатися і наступного дня після роботи з комп'ютером. Жінки частіше, ніж чоловіки, скаржаться на зоровий дискомфорт при роботі за комп'ютером. У жінок у віці 31-45 років астенопія виникає частіше, ніж у жінок 18-30 років, що свідчить про вплив стажу. Астенопія часто зустрічається у операторів з введення даних, прийому даних і діалогової роботи. До 47% користувачів скаржаться на зорове стомлення, якщо робота без перерви триває менше 30 хв, і 66% — при безперервній роботі більше 30 хв. Очні симптоми більш виражені у тих осіб, що менше контролюють свою роботу, працюють з великим напруженням і відчувають незадоволеність своєю працею. Більш чутливими до виникнення астенопії є люди з порушеннями зору.

Тривала робота у вимушеній робочій позі сидячи призводить до довготривалого статичного напруження одних і тих самих груп м'язів. Часто користувачі ЕОМ змушені сидіти у сутулій позі, яка характеризується максимальним згинанням у поперековому і грудному відділах хребта та нахилом тулуба вперед. Ця поза має низку несприятливих біомеханічних і фізіологічних наслідків, що призводить до виникнення болю у шийних і поперекових відділах хребта, остеохондрозу хребта, геморою , простатиту та інших захворювань . У підлітків під час тривалої роботи з комп'ютером може виникати сколіоз.

При виконанні дрібних стереотипних рухів під час роботи з клавіатурою і мишею, м'язи рук користувачів знаходяться в постійному напруженні, як наслідок — виникнення синдрому зап'ясткового каналу.

Перевантаження певних ґруп м’язів під час роботи на комп’ютері поєднується з гіпокінезією. При зниженні рівня фізичної активності різко збільшується ризик виникнення багатьох інших захворювань.

Окремо зупинимося на негативному впливі на здоров’я людини змінних електромагнітних полів (ЕМП), зокрема радіочастотного випромінювання при роботі моніторів на основі електронно-променевої трубки (ЕПТ).

Фахівцями Рівненської обласної санітарно-епідеміологічної станції ДСЕС України та Рівненського обласного лабораторного центру МОЗ України протягом 2009-2015 років проводились виміри електромагнітних випромінювань від персональних комп’ютерів в школах області. Досліджено 1171 ПК, біля яких зроблено 2313 вимірів електромагнітного випромінювання, з яких не відповідало нормам 635 вимірів (27,4%). Невідповідності нормам (перевищення до 2-х разів) були зафіксовані біля відеотерміналів на основі ЕПТ, в складі ПК, які не були підключені до контуру заземлення електричного обладнання. Після усунення такого недоліку, повторні виміри рівнів ЕМП не зафіксували відхилень від норми.

Дія електромагнітного випромінювання широкого спектру частот, імпульсного характеру та різної інтенсивності при роботі ЕОМ, у поєднанні з високим зоровим і нервово-емоційним напруженням викликає істотні зміни з боку центральної нервової і серцево-судинної систем, які проявляються у суб’єктивних і об’єктивних розладах. Працюючі з комп’ютером особи найчастіше висловлюють скарги на головний біль, іноді з нудотою і запамороченням. У них частіше, ніж у осіб контрольних груп, діагностуються неврози, нейроциркуляторні дистонії, гіпо- і гіпертонії.

ВИСНОВКИ

Кваліфікаційна робота виконана з метою вивчення, розробки та впровадження системи автоматизованого тестування для забезпечення якості програмного продукту. В процесі дослідження предметної області було проведено огляд існуючих засобів автоматизованого тестування, визначено поняття та переваги автоматизованого тестування.

Основний акцент був зроблений на детальному описі розробки та конфігурації системи автоматизованого тестування у середовищі Intellij IDEA. Вибір інтегрованого середовища розробки був обгрунтований зручністю роботи з інструментами та підтримкою мови програмування Java.

Подальший аналіз і розробка дозволили підвищити якість та надійність програмного продукту. Використання Page Object паттерну, модульного підходу та інших передових практик сприятило підтримуваності та розширюваності тестового набору.

Інтеграція з Jenkins дозволила автоматизувати процес Continuous Integration, що важливо для забезпечення швидкості та ефективності розробки. Використання засобів візуалізації результатів та логів, зокрема ExtentReport, полегшило аналіз.

Спроектоване програмне забезпечення виокремило ключові аспекти автоматизованого тестування, надало практичний досвід розробки та впровадження системи, а також вказало на перспективи подальшого вдосконалення та розвитку даної тематики.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Automation Testing [Електронний ресурс] // Режим доступу: https://www.javatpoint.com/automation-testing
2. Selenium Getting Started [Електронний ресурс] // Режим доступу: https://www.selenium.dev/documentation/webdriver/getting_started/
3. Selenium WebDriver Tutorial : Getting Started with Test Automation [Електронний ресурс] // Режим доступу: https://www.browserstack.com/guide/selenium-webdriver-tutorial
4. Building Java Projects with Gradle [Електронний ресурс] // Режим доступу: https://spring.io/guides/gs/gradle/
5. IntelliJ IDEA – Getting started [Електронний ресурс] // Режим доступу: https://www.jetbrains.com/help/idea/getting-started.html
6. IntelliJ IDEA – Pro tips [Електронний ресурс] – Режим доступу: https://www.jetbrains.com/help/idea/pro-tips.html
7. Learn Selenium with Java to run Automated Tests [Електронний ресурс] // Режим доступу: https://www.browserstack.com/guide/selenium-with-java-for-automated-test
8. Java for Testers [Електронний ресурс] // Режим доступу: https://artoftesting.com/java-for-testers
9. Jenkins - Getting started with the Guided Tour [Електронний ресурс] // Режим доступу: https://www.jenkins.io/doc/pipeline/tour/getting-started/
10. Jenkins - Getting started with Pipeline [Електронний ресурс] // Режим доступу: https://www.jenkins.io/doc/book/pipeline/getting-started/

ДОДАТКИ

Додаток А

Тези конференції

УДК 004.41

А.В. Мельник

Тернопільський національний технічний університет імені Івана Пулюя, Україна

Система автоматизованого тестування з використанням інструментів Selenium і Jenkins та середовища Intellij IDEA

A.V. Melnyk

AUTOMATED TESTING SYSTEM USING SELENIUM AND JENKINS TOOLS AND INTELLIJ IDEA ENVIRONMENT

Сучасні технології розробки програмного забезпечення вимагають від розробників високої якості коду та ефективного використання ресурсів. Одним з ефективних методів забезпечення цих вимог є впровадження системи автоматизованого тестування. Метою є розробка та інтеграція системи автоматизованого тестування, заснованої на інструментах Selenium і Jenkins, у популярному середовищі розробки Intellij IDEA.

Робота передбачає ретельний аналіз основних проблем, які виникають під час розробки програмного забезпечення, і визначення ключових вимог до системи тестування. Застосування інструментів Selenium для автоматизації тестування взаємодії користувача з веб-додатками дозволить ефективно перевіряти функціональність та стабільність програми.

Додатково, інтеграція інструменту Jenkins у систему дозволить автоматизувати процес тестування та забезпечити постійний моніторинг стабільності розроблюваного програмного продукту. Особлива увага буде приділена налагодженню взаємодії між інструментами та їх оптимальному конфігуруванню для досягнення найвищої ефективності та продуктивності.

Підсумковою частиною роботи буде проведення експериментальних випробувань розробленої системи на реальному проекті, а також аналіз отриманих результатів. Реалізація даної системи дозволить підвищити якість розробки програмного забезпечення, зменшити час, необхідний для виявлення та виправлення помилок, та забезпечить більш ефективне використання ресурсів розробників.